Lección 3.4 --- Desarrollo de software seguro

Desarrollo de software seguro

Desarrollo de software seguro

Algunas de las prácticas de codificación más importantes son la validación de entrada, la codificación de salida y el manejo de errores.

Validación de entradas

Un vector principal para atacar aplicaciones es explotar la validación de entrada defectuosa. La entrada podría incluir datos de usuario ingresados en un formulario o URL pasados por otra aplicación como URL o encabezado HTTP. La entrada maliciosa podría diseñarse para realizar un ataque de desbordamiento o algún tipo de secuencia de comandos o ataque de inyección SQL. Para mitigar este riesgo, todos los métodos de entrada deben documentarse con miras a reducir la superficie de ataque potencial expuesta por la aplicación. Debe haber rutinas para verificar la entrada del usuario, y cualquier cosa que no se ajuste a lo requerido debe ser rechazada.

Normalización y codificación de salida

Cuando una aplicación acepta la entrada de cadenas, esta debe someterse a procedimientos de normalización antes de ser aceptada. La normalización significa que en una cadena se eliminan los caracteres o subcadenas ilegales y se convierte al conjunto de caracteres aceptado. Esto garantiza que la cadena esté en un formato que las rutinas de validación de entrada puedan procesar correctamente. Cuando las cadenas generadas por el usuario se pasan a través de diferentes contextos en una aplicación web, entre HTTP, JavaScript, PHP y SQL, por

ejemplo, cada uno con esquemas de canonicalización potencialmente diferentes, es extremadamente difícil garantizar que los caracteres que facilitaríian la inyección de script por XSS se hayan protegido correctamente.

La codificación de salida significa que la cadena se vuelve a codificar de forma segura para el contexto en el que se utiliza. Por ejemplo, un formulario web puede realizar la validación de entrada en el cliente, pero cuando llega al servidor, una función de PHP realiza la codificación de salida antes de escribir una declaración SQL. De manera similar, cuando se entrega una cadena desde una base de datos usando SQL, una función de JavaScript realizaría la codificación de salida para representar la cadena usando entidades HTML seguras.

Manejo de errores

Una aplicación bien escrita debe ser capaz de manejar errores y excepciones de manera adecuada. Esto significa que la aplicación funciona de forma controlada cuando sucede algo impredecible. Un error o una excepción puede deberse a una entrada de usuario no válida, una pérdida de conectividad de red, una falla en el proceso o en el servidor, etc. Idealmente, el programador habrá escrito un controlador de excepción estructurado (SEH) para indicar qué debe hacer la aplicación. Cada procedimiento puede tener varios controladores de excepciones. Algunos controladores se ocuparán de los errores y excepciones previstos; también debe haber un controlador general que se ocupe de lo inesperado. El objetivo principal debe ser que la aplicación no falle de forma que permita al atacante ejecutar código o realizar algún tipo de ataque de inyección. Otro problema es que el intérprete de una aplicación puede usar de forma predeterminada un controlador estándar y mostrar mensajes de error predeterminados cuando algo sale mal. Estos pueden revelar información de la plataforma y el funcionamiento interno del código a un atacante. Es mejor que una aplicación utilice controladores de errores personalizados

para que el desarrollador pueda elegir la cantidad de información que se muestra cuando se produce un error.

Uso de código seguro

Desarrollar código para realizar alguna función es un trabajo arduo, por lo que los desarrolladores a menudo buscarán si alguien más ya ha hecho ese trabajo. Un programa puede hacer uso del código existente de las siguientes maneras:

- Reutilización de código: uso de un bloque de código de otra parte de la misma aplicación o de otra aplicación para realizar una función diferente (o realizar la misma función en un contexto diferente). El riesgo aquí es que el enfoque de copiar y pegar hace que el desarrollador pase por alto vulnerabilidades potenciales (tal vez los parámetros de entrada de la función ya no se validen en el nuevo contexto).
- Librerías de terceros: utilizar un paquete binario que implementa algún tipo de funcionalidad estándar, como establecer una conexión de red o implementar criptografía. Cada librería debe ser monitoreada en busca de vulnerabilidades y parchada de inmediato.
- Kit de desarrollo de software (SDK): utiliza código de muestra o librerías de funciones preconstruidas del entorno de programación utilizado para crear el software o interactuar con una API de terceros. Al igual que con otras bibliotecas o códigos de terceros, es necesario monitorear las vulnerabilidades.
- Procedimientos almacenados: uso de una función preconstruida para realizar una consulta de base de datos. Un procedimiento almacenado es una parte de una base de datos que ejecuta una consulta personalizada. El programa de llamada proporciona una entrada al procedimiento y devuelve una salida predefinida para los registros coincidentes. Esto puede proporcionar un medio más seguro para consultar la base de datos. Cualquier procedimiento almacenado que forme parte de la base de datos pero que la aplicación no requiera debe estar deshabilitado.

Otras prácticas de codificación segura

El manejo de entrada y error, además de la reutilización segura del código existente, cubre algunas de las principales prácticas de desarrollo relacionadas con la seguridad que se deben tener en cuenta. Hay algunos otros problemas que pueden surgir durante el desarrollo y la implementación del código de la aplicación. Ejemplo de esto son el código inalcanzable y el código muerto.

El código inalcanzable es una parte del código fuente de la aplicación que nunca se puede ejecutar. Por ejemplo, puede haber una rutina dentro de una declaración lógica (Si... Entonces) que nunca se puede llamar porque nunca se pueden cumplir las condiciones que la llamarían. Por otro lado, el código muerto se ejecuta pero no tiene efecto en el flujo del programa. Por ejemplo, puede haber código para realizar un cálculo, pero el resultado nunca se almacena como una variable ni se usa para evaluar una condición. Este tipo de código puede introducirse a través de código reutilizado sin cuidado, o cuando se reescribe o cambia un bloque de código. El código inaccesible y muerto debe eliminarse de la aplicación para evitar la posibilidad de que pueda ser mal utilizado de alguna manera. La presencia de código inalcanzable/muerto puede indicar que la aplicación no se está manteniendo bien.

Generalmente es importante que el código esté bien documentado para facilitar el trabajo a los programadores que trabajan en el mismo proyecto. Sin embargo, el código bien documentado también es más fácil de analizar, lo que puede ayudar al desarrollo de ataques. El código puede resultar difícil de analizar mediante el uso de un ofuscador, que es un software que aleatoriza los nombres de variables, constantes, funciones y procedimientos, elimina comentarios y espacios en blanco, y realiza otras operaciones para hacer que el código sea difícil de leer y seguir. Este tipo de técnica podría usarse para hacer que la ingeniería inversa sea más difícil.